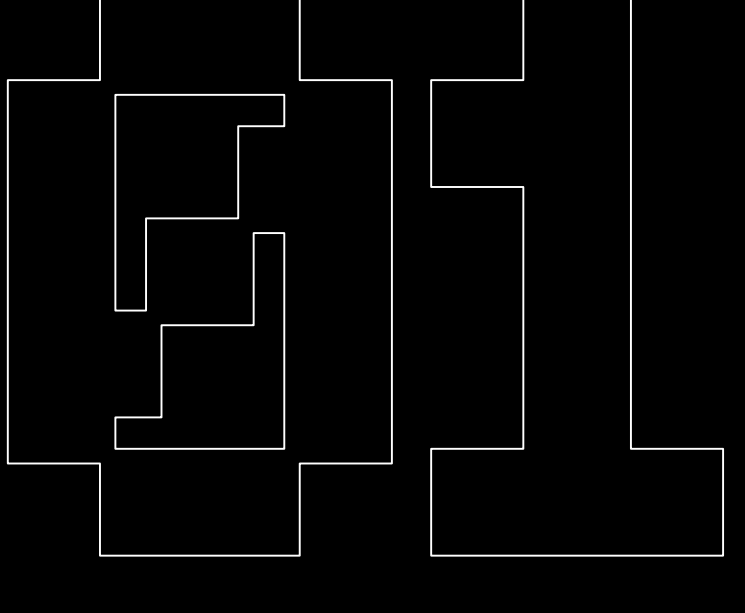
PIXEL POWER

PROGRAMANDO O FUTURO DOS GAMES

SIMONE PAIVA

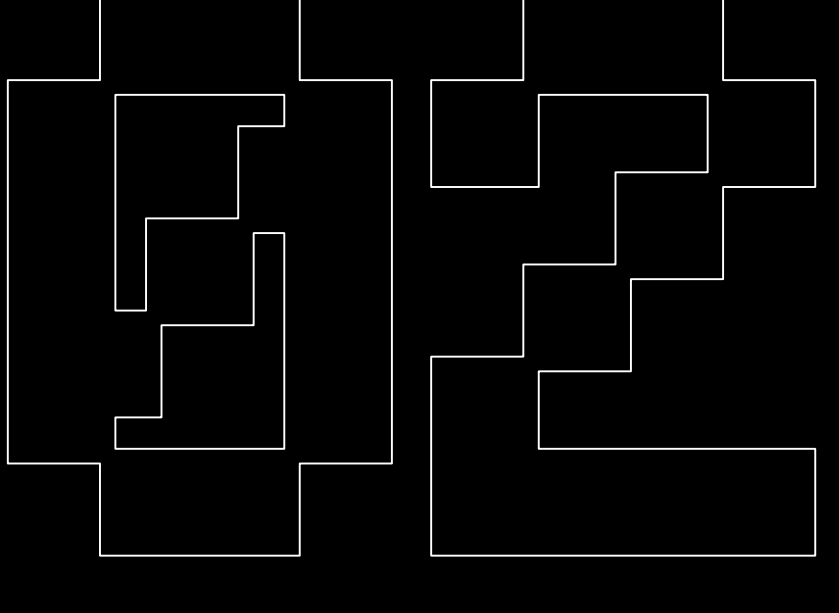


Estruturas de Dados: Gerenciando Entidades no Jogo

. C++ é uma das linguagens mais poderosas para o desenvolvimento de jogos, sendo amplamente utilizada em motores gráficos e para criar jogos de alto desempenho. A seguir, vamos explorar os principais setores da programação de jogos com exemplos simples, para te ajudar a entender como o C++ pode ser usado na prática.

Lista Encadeada de Inimigos

```cpp
struct Enemy {
    int id;
    Enemy* next;
};

class EnemyList {
public:
    Enemy* head;

    EnemyList() : head(nullptr) {}

    void addEnemy(int id) {
        Enemy* newEnemy = new Enemy{id, nullptr};
        if (!head) head = newEnemy;
        else {
            Enemy* temp = head;
            while (temp->next) temp = temp->next;
            temp->next = newEnemy;
        }
    }

    void printEnemies() {
        Enemy* temp = head;
        while (temp) {
            std::cout << "Inimigo ID: " << temp->id << std::endl;
            temp = temp->next;
        }
    }
};
```

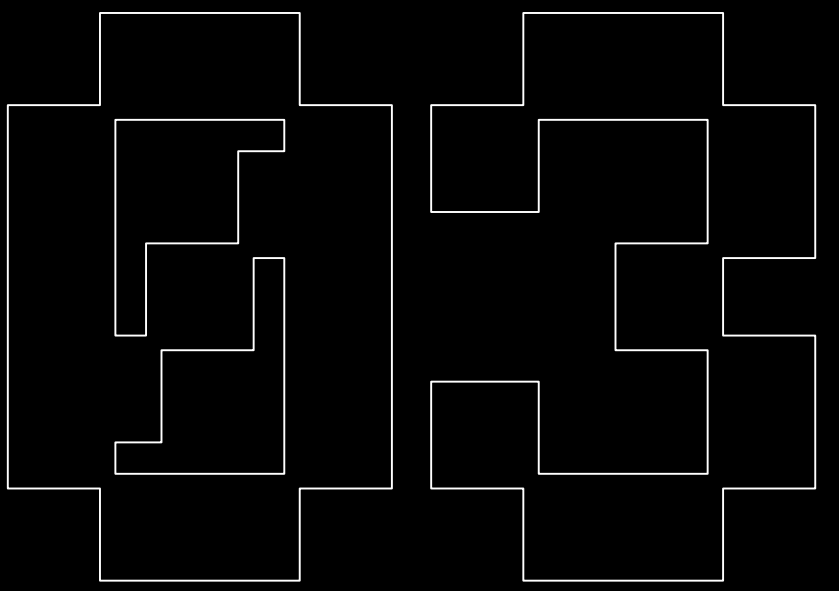


Controlando Objetos Dinâmicos

. Em jogos, você cria e destrói muitos objetos em tempo real. O C++ exige que você gerencie manualmente a memória, o que permite otimizar o uso de recursos.

Alocação Dinâmica de Objetos

```cpp
class Player {
public:
    Player() { std::cout << "Jogador Criado!" << std::endl; }
    ~Player() { std::cout << "Jogador Destruído!" << std::endl; }
};

int main() {
    Player* player = new Player();  // Criação do jogador
    delete player;  // Liberação da memória
    return 0;
}
```

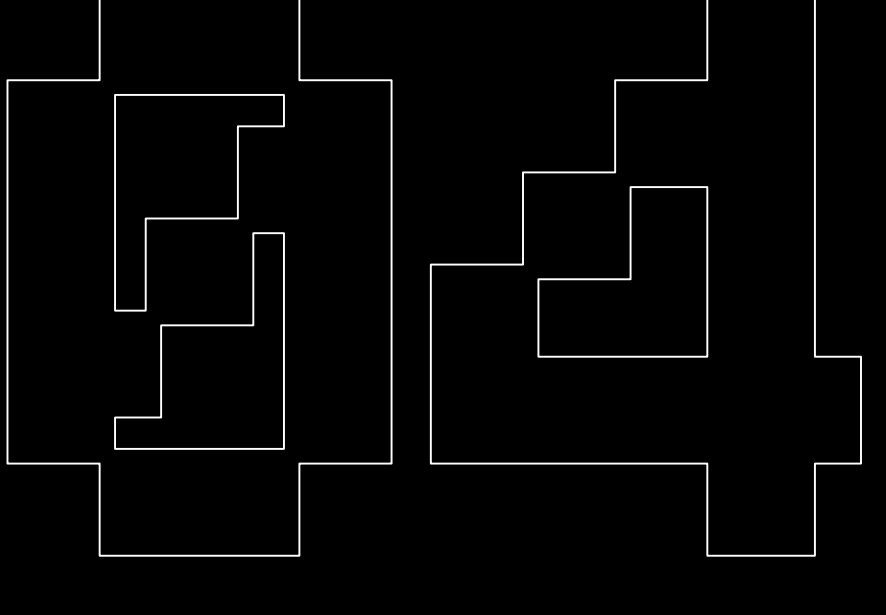


Exibindo Imagens na Tela

C++ é frequentemente usado com bibliotecas gráficas como OpenGL ou SFML para renderizar gráficos 2D e 3D.

Desenhando um Triângulo com OpenGL

```cpp
#include <GL/glut.h>

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
        glVertex2f(-0.5f, -0.5f);
        glVertex2f( 0.5f, -0.5f);
        glVertex2f( 0.0f,  0.5f);
    glEnd();
    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutCreateWindow("Jogo C++");
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

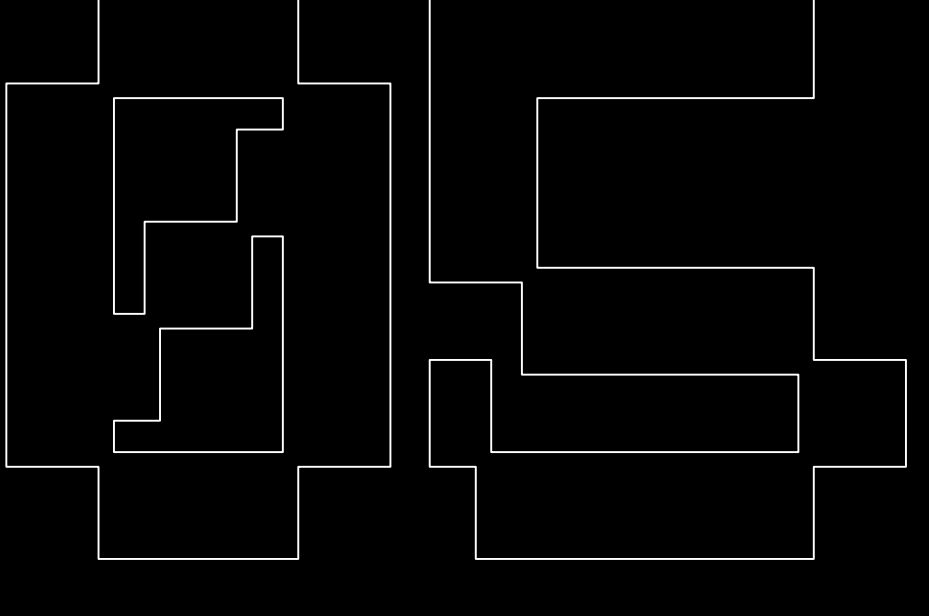


Simulando Movimento Realista

A física nos jogos envolve simular o movimento de objetos, incluindo forças como gravidade e colisões.

Simulando Gravidade

```cpp
class Object {
public:
    float x, y, vx, vy;

    void update(float dt) {
        vy -= 9.8f * dt;  // Gravidade
        x += vx * dt;
        y += vy * dt;
    }
};

int main() {
    Object player;
    player.x = 0.0f;
    player.y = 10.0f;
    player.vx = 0.0f;
    player.vy = 0.0f;

    player.update(0.016f);  // Atualização de física (simula 60fps)

    std::cout << "Posição do jogador: (" << player.x << ", " << player.y << ")\n";
    return 0;
}
```

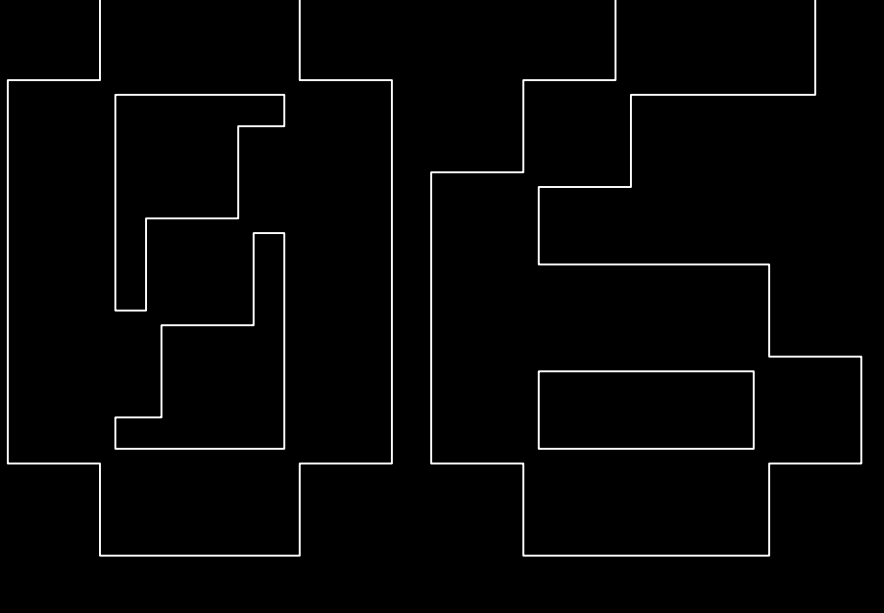


Inteligência Artificial (IA): Criando Comportamento dos Inimigos

. Em jogos, a IA controla o comportamento dos personagens não-jogáveis (NPCs). No C++, podemos usar máquinas de estados ou algoritmos de busca como A*.

Máquina de Estado para Inimigos

```cpp
enum State { CHASE, FLEE, IDLE };

class Enemy {
public:
    State state;

    void update() {
        if (state == CHASE) std::cout << "Inimigo Perseguindo\n";
        else if (state == FLEE) std::cout << "Inimigo Fugindo\n";
        else std::cout << "Inimigo Inativo\n";
    }
};

int main() {
    Enemy enemy;
    enemy.state = CHASE;
    enemy.update();  // Inimigo começa a perseguir

    return 0;
}
```

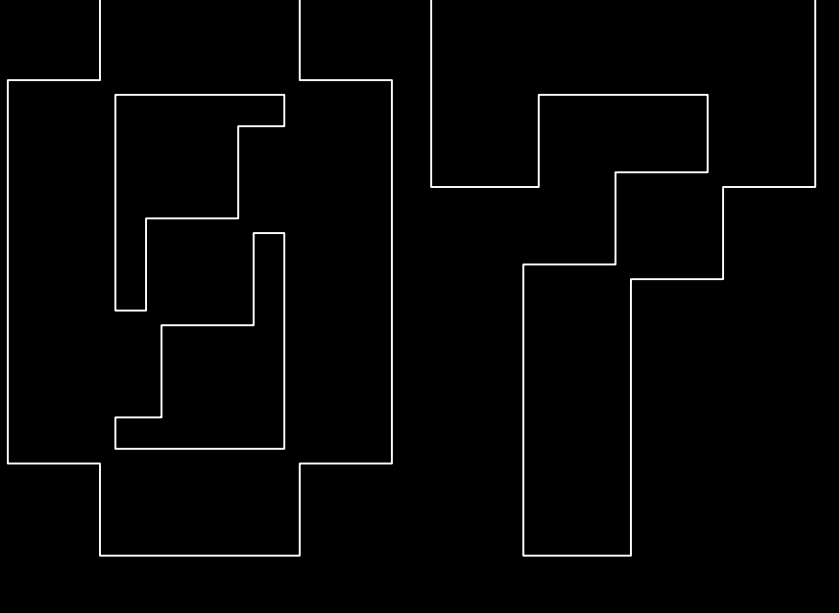


Multiplayer: Conectando Jogadores

Jogos multiplayer exigem comunicação entre jogadores. Usamos sockets para enviar e receber dados de rede.

Enviando Mensagens entre Cliente e Servidor

```
#include <winsock2.h>
#include <iostream>

int main() {
    WSADATA wsaData;
    SOCKET sock;
    sockaddr_in server;

    WSAStartup(MAKEWORD(2, 2), &wsaData);
    sock = socket(AF_INET, SOCK_STREAM, 0);
    server.sin_family = AF_INET;
    server.sin_port = htons(8888);
    server.sin_addr.s_addr = inet_addr("127.0.0.1");

    connect(sock, (sockaddr*)&server, sizeof(server));
    const char* message = "Jogador Conectado!";
    send(sock, message, strlen(message), 0);

    closesocket(sock);
    WSACleanup();

    return 0;
}
```



Áudio: Sons e Música no Jogo

Adicionar áudio é crucial para a imersão do jogador. Em C++, podemos usar bibliotecas como SFML para reproduzir sons.

Reproduzindo um Efeito Sonoro

```cpp
#include <SFML/Audio.hpp>

int main() {
    sf::SoundBuffer buffer;
    if (!buffer.loadFromFile("som.ogg")) {
        std::cerr << "Erro ao carregar som!" << std::endl;
        return -1;
    }

    sf::Sound sound;
    sound.setBuffer(buffer);
    sound.play();  // Reproduz o som

    // Espera o som terminar
    while (sound.getStatus() == sf::Sound::Playing) {}

    return 0;
}
```

Conclusões

C++ oferece ferramentas poderosas para criar jogos de alto desempenho. Com exemplos práticos de estruturas de dados, gerenciamento de memória, gráficos, física, IA, multiplayer e áudio, você tem as bases para começar a desenvolver jogos incríveis. A chave para o sucesso é entender esses conceitos e aplicá-los em cenários reais, como os exemplos acima, adaptando-os às necessidades do seu jogo.